

---

# HDF4 Mapping Prototype

## Design and Schema

---

Peter Cao ([xcao@hdfgroup.org](mailto:xcao@hdfgroup.org))

Ruth Aydt ([aydt@hdfgroup.org](mailto:aydt@hdfgroup.org))

The HDF Group  
September 17, 2008

*A desire to read HDF4 files without relying on the HDF4 library prompted the work described in this document—the construction of text-based “maps” of binary HDF4 files.*

*An XML-based prototype schema for HDF4 mapping files (XML documents) was created. For a given binary HDF4 file, an associated mapping file contains structural and application metadata for the HDF4 file, as well as the locations of the object data (array element values) in the HDF4 file. A tool was written to generate mapping files. Other tools were developed that use the mapping files to read HDF4 files without calling the HDF4 library, confirming the approach is viable.*

*While the focus of this effort was NASA EOSDIS data stored in HDF4 files, the general methodology is also relevant to other cases where the long-term accessibility of data stored in binary files is of concern. In addition, this work demonstrates how binary HDF files can be used to efficiently store large volumes of scientific data that is referenced by text-based XML documents (the mapping files).*

*Part One of this document provides an overview of the HDF4 mapping design considerations. Part Two presents the prototype HDF4 mapping schema and sample mapping files.*

---

## Part One: HDF4 Mapping Prototype Design Considerations

---

Part One introduces the HDF4 mapping prototype project and describes the general design considerations.

### 1.1 Introduction

---

The Hierarchical Data Format (HDF) has been a data format standard in NASA's Earth Observing System Data and Information System (EOSDIS) since the 1990s. HDF's rich structure, platform independence, full-featured Application Programming Interface (API), and internal compression make it well-suited for science data that is shared and accessed via a rich set of software tools. HDF files are *self-describing*; that is, for each HDF data structure in an HDF file there is comprehensive information about the data and its location in the file.

The internal byte layout of HDF files, which was designed for high-performance and space-efficiency, is quite complex. This complexity is a major drawback for long-term data archival, as it practically

necessitates the use of the HDF API to access the data. Reliance on the HDF API makes the long-term readability of HDF data dependent on recurring resource allocations to support the HDF library. Different versions of the HDF format have different internal byte layouts and APIs. There are currently two versions of the Hierarchical Data Format—Version 4 (HDF or HDF4) and Version 5 (HDF5). The majority of the data from NASA’s Earth Observing System (EOS) has been archived in the HDF Version 4 format.

To begin to address the long-term archival issues for the NASA-held HDF4 data, a collaborative study between The HDF Group and NASA’s EOSDIS data centers was carried out. This document and the work it presents are products of that project.

One of the first activities undertaken by the project was an assessment of the range of HDF4-formatted data held by NASA. The assessment helped identify the capabilities inherent in the HDF4 format that have been used in practice. It also helped quantify the amount of data NASA has stored in the HDF4 format, an important factor when estimating the effort it will take to fully implement any approach to alternative access methods across NASA’s EOSDIS data centers.

Based on the results of this assessment, the project defined a prototype schema for text-based files that “map” the contents of the HDF Version 4 files held by NASA. As part of the project, a prototype tool, *hmap*, was written to generate text-based *mapping files* from HDF4 files. Prototype reader programs were written in C and Perl. The ability of the reader programs to retrieve object data from a variety of NASA’s data products using the information contained in the mapping files, and without reliance on the HDF4 library, verifies the mapping approach is viable. Furthermore, the HDF4 mapping design makes the object data stored in HDF4 files accessible without requiring that it be rewritten to another format—a potentially time-consuming and costly proposition.

The paper “Ensuring Long Term Access to Remotely Sensed Data with Layout Maps”, authored by R. Duerr, P. Cao, J. Crider, M. Folk, C. Lynnes, and M. Yang, provides a more in-depth treatment of the information highlighted in this section. It is currently in press and scheduled to appear in *IEEE Transactions on Geoscience and Remote Sensing (TGARS)*, special issue on *Data Archiving and Distribution*, 2008.

## 1.2 HDF-EOS Data Objects and Metadata

---

To reflect the nature of the data being collected and stored, and to provide a consistent and intuitive interface to the consumers of its data products, NASA has developed conventions for storing data from the EOS program. HDF-EOS files are HDF4 files that adhere to those conventions. The HDF-EOS software library implements the HDF-EOS conventions, and facilitates interactions with HDF-EOS files.

HDF-EOS data objects include *points*, *swaths*, and *grids*. HDF-EOS metadata includes information needed to interpret the HDF-EOS data such as ESDT, level, product name, swath data organization (time, space, both), grid data # of projections, and so on.

HDF-EOS data objects and HDF-EOS metadata are stored as HDF4 objects and HDF4 metadata. There is not a one-to-one mapping between the HDF-EOS and HDF4 representations. The “translations” between HDF4 and HDF-EOS representations are made by the HDF-EOS library, which presents applications calling the library with the data and metadata in the HDF-EOS format. Some of the translations, such as the construction of the grid, point, and swath data objects from the more generic HDF4 objects and metadata, require domain knowledge and can be quite complex.

## 1.3 HDF4 Objects, Object Data, and Metadata

---

The HDF4 format supports data structures that store and organize data values. These data structures include *Vdatas* (tables), *Scientific Data Sets* (multi-dimensional arrays), and *raster images*. In this document,

an instance of one of these HDF4 data structures is called an HDF4 *data object*. The data values that are stored in the underlying array elements of these HDF4 data objects are referred to as *object data*.

For improved performance and space efficiency, the HDF4 library can apply transformations to the object data prior to storing it in an HDF4 file. Any application that reads the HDF4 file without using the HDF4 library must be able to reverse the transformations that were applied to the object data when it was stored. The transformations include subdividing and compression.

The HDF4 *Vgroup* structure can be used to group data objects, and to create hierarchies of objects. An instance of the *Vgroup* structure is called a *grouping object* in this document.

Metadata can be associated with specific data and grouping objects, as well as with an entire HDF4 file. Some of this metadata, *structural metadata*, is stipulated by the HDF4 format and used to fully specify characteristics of the objects. Other metadata, *application metadata*, is information added by, and meaningful to, the application writing or reading the files.

The HDF4 format also includes other structures, but the ones mentioned here are the most critical for the mapping prototype project. *The mapping prototype project focused on the HDF4 objects that were identified as being most important in the NASA EOSDIS data.*

#### 1.4 Levels of Information: HDF-EOS and HDF4 Mapping

---

The information in an HDF-EOS file was conceptually divided into three levels during the design phase of the mapping prototype project. The three levels are shown in Figure 1. Levels I and II are HDF4-specific. Level III is HDF-EOS specific.

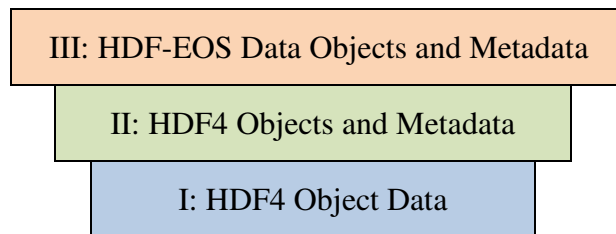


Figure 1 – Levels of information in an HDF-EOS file.

The study examined information at all three levels. After careful consideration, the decision was made to define elements in the XML-based prototype mapping schema corresponding to the HDF4-specific information (Levels I and II), but not to the HDF-EOS information (Level III). This design decision resulted in prototype mapping files whose elements are relevant to all HDF4 files.

The Level III HDF-EOS information is represented as Level I and Level II HDF4 information in the mapping files; there is no direct representation of the HDF-EOS data objects and metadata. Tools tailored to the HDF-EOS project can access and interpret the Level III information encapsulated in the HDF4 mapping files, and perform the translations from the HDF4 objects and metadata to the HDF-EOS objects and metadata. This is similar to the functionality the HDF-EOS library provides for the original HDF4 files.

A further exposition of the Level I and Level II mapping information follows.

##### 1.4.1 Level I: HDF4 Object Data

---

For a typical HDF4 file, the majority of the data in the file is object data—the data values in the *Vdata*, Scientific Data Set (SDS), and raster image data objects. The volume of the object data makes it impractical, both in terms of time and space, to rewrite the data values in text-based files. Because of

these considerations, the mapping project leaves the object data in the original (binary) HDF4 file, and provides the details needed to retrieve the values in an associated text-based HDF4 mapping file that can be archived along with the original HDF4 file.

For each HDF4 data object, the mapping file must include the offset and size (number of bytes) of the object data in the HDF4 file. When applicable, subdivision and compression details needed to reverse transformations are also included. A mapping file does not contain any object data—it only contains the locations of the object data in the original HDF4 file and the pertinent information about transformations.

Section 1.5 provides additional details about the storage layout of object data in binary HDF4 files. Part II provides details on how XML is used to represent the Level I object data location and transformation information in the prototype HDF4 mapping file.

#### *1.4.2 Level II: HDF4 Objects and Metadata*

---

Level II information can be thought of as everything in an HDF4 file that 1) is not object data and 2) is not based on a data model for an application domain. Just as a road map has a legend and labels that are critical to its meaning, Level II information is essential to making sense of the object data in an HDF4 file. Much of what makes HDF4 self-describing is the Level II information.

Level II information includes:

- The object hierarchy and grouping conveyed by Vgroup objects in the HDF4 file.
- The number and type (SDS, Vtable, raster image) of data objects.
- Structural metadata (e.g., name, data type, byte order, number and size of dimensions, palette).
- Application metadata (annotations and attributes).

The decision was made to directly represent all Level II information in the HDF4 mapping files for three primary reasons:

1. Level II information is critical to understanding an HDF4 file, and for some applications is all that is needed.
2. The data volume associated with Level II information is typically small in comparison to that of the Level I object data.
3. Level II information can be readily represented as text.

### *1.5 HDF4 Object Data Storage Layout*

---

A brief introduction to the storage layout of the object data in binary HDF4 files is presented in this section. A detailed description is given in *The HDF4 Specification and Developer's Guide*, which can be found at <http://www.hdfgroup.org/doc.html>.

The object data (i.e., contents of a data array) in an HDF4 data object can be stored as a single contiguous block, as linked blocks, or as chunks in the HDF4 file. These blocks and chunks can be compressed by DEFLATE, SZIP, JPEG, or N-bit run-length encoding algorithms. An HDF4 data object contains information about where its object data is stored in the HDF4 file. A data offset field contains the number of bytes from the beginning of the HDF4 file to the beginning of the object data. A length field contains the size of the object data in bytes.

With the *contiguous* storage layout, the object data is stored in the file as a single data block. An <offset, length> pair is sufficient to retrieve the object data from the HDF4 file. Figure 2 shows a simplified example of the contiguous storage layout, where the offset is 2502 and the length is 2400.

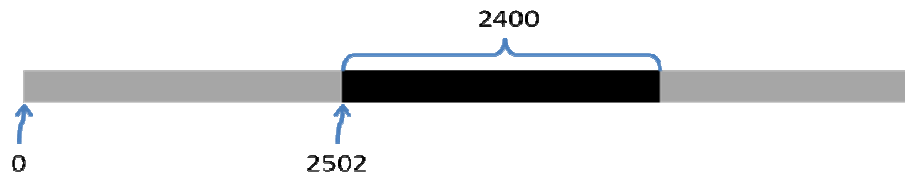


Figure 2 – Simplified example of contiguous storage layout.

The *linked blocks* storage layout uses a series of data blocks spread throughout the file that are chained together in a linked list. Linked blocks storage provides a convenient means of growing the object data without rewriting existing data. To retrieve object data stored in linked blocks, the offset and length of each linked block is needed.

With the *chunked* storage layout, the raw data is divided into equally-sized segments (chunks) that are stored and retrieved separately. A chunk is a hyper-rectangle of any shape. The number of dimensions in a chunk must be equal to the number of dimensions in the data array. To read chunked data from a file, the coordinates of each chunk in the overall chunk array, as well as the offset and length of each chunk are needed. This information is stored in a chunk table in the HDF4 file. Figure 3 shows a simplified example of a chunked storage layout.

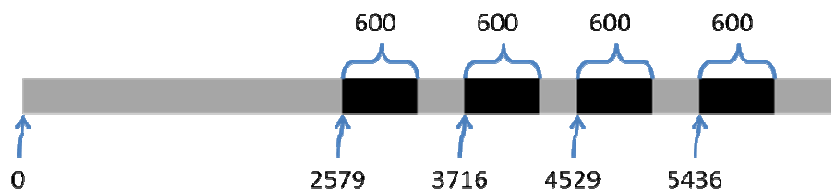


Figure 3 – Simplified example of chunked storage layout.

Object data can be compressed with a compression algorithm such as DEFLATE, SZIP, JPEG, or N-bit run-length encoding. When the chunked storage layout is used, compression is performed on a chunk-by-chunk basis. After compression, the size of each chunk may be different. Figure 4 shows a simplified example of compressed data with the chunked storage layout.

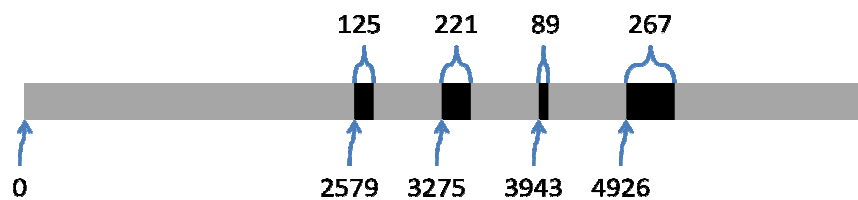


Figure 4 – Simplified example of compressed chunks.

The HDF4 mapping file must contain sufficient information about the object data storage layout, <offset, length> pair(s), and compression method/parameters to allow an application to locate the object data in the binary HDF4 file and uncompress it—without relying on the HDF4 library.

As an example of what is possible when the compression information is known, Figure 5 presents a C function that will uncompress HDF4 object data that was compressed using the DEFLATE algorithm. The function, *deflate\_decode*, takes several parameters: *level* is the compression level, *buf\_in* points to the buffer holding the compressed data that is *len\_in* bytes long, and *buf\_out* points to the buffer for the uncompressed data that is *len\_out* bytes long.

Section 2.2.9 discusses how the location, storage layout, and transformation information is represented in the HDF4 mapping files.

```
static int
deflate_decode(int level, int len_in, uint8 *buf_in, int len_out, uint8 *buf_out)
{
    int      status;          /* inflate status */
    z_stream zs;             /* deflation context */
    int32    bytes_out;

    memset(&zs, 0, sizeof(zs));

    zs.next_in = buf_in;
    zs.avail_in = (uInt)len_in;
    zs.next_out = buf_out;
    zs.avail_out = (uInt)len_out;

    if (Z_OK!=inflateInit(&zs)) {
        puts("inflateInit() failed");
        return -1;
    }

    do {
        /* Read compressed data */
        status=inflate(&(zs),Z_SYNC_FLUSH);
    } while (status==Z_OK);

    (void)inflateEnd(&zs);
    bytes_out=(int32)len_out-(int32)zs.avail_out;

    return(bytes_out);
} /* end deflate_decode() */
```

Figure 5 – C function to uncompress data originally compressed with the DEFLATE algorithm.

## 1.6 Prototype Mapping Tools

---

The project developed two types of prototype tools: an HDF4 mapping file generator and multiple HDF4 mapping/data file readers.

The HDF4 mapping file generator, *hmap*, creates a mapping file for a given HDF4 file based on the mapping schema presented in Part Two. The generator is HDF4-specific, and handles Level I and Level II information. Since HDF4 is a very general format with considerable variation in data objects and the way they are stored, this study focused on the basic HDF4 data products produced by NASA. *It was beyond the scope of this project to handle all possible HDF4 files.*

The HDF4 mapping/data file readers use the information in an HDF4 mapping file to retrieve object data from the associated HDF4 file. Prototype reader programs were written in C and Perl. The ability of the reader programs to retrieve object data from a variety of NASA's data products verifies the mapping approach is viable. The readers do not depend on the HDF4 library.

## Part Two: Prototype Mapping Schema and Sample Mapping Files

---

Part Two presents the prototype HDF4 mapping schema and sample mapping files.

### 2.1 Why XML?

---

To satisfy the goal of retrieving data from an HDF4 file without using the HDF4 library, and to ensure that the new approach does not introduce comparable levels of complexity, the mapping files should be human-readable, portable, standard, and independent of any binary format.

XML is the most suitable mapping file format for several reasons:

- XML documents are plain text and do not depend on any binary format.
  - This feature is very important for data archiving because users must be able to read mapping information with any standard text editor.
- XML documents are machine-readable.
  - This feature makes writing applications to read the documents more straightforward.
  - Numerous tools exist for checking the syntax and validity of the documents produced.
- XML documents are hierarchical.
  - An XML document can easily and naturally reflect the structure of an HDF4 file.
- XML schemas exist for HDF5 and HDF-EOS 5.
  - By writing the HDF4 mapping file in XML, the HDF4 mapping schema can be extended and applied to HDF5 and HDF5-EOS 5 files in the future.
- Other schema for data archiving, such as the PREMIS preservation metadata schema, are written in XML.
  - Using XML for the prototype HDF4 mapping files will help with the adoption of other XML-based preservation schema in the future.

### 2.2 Prototype XML Schema

---

The prototype XML schema defines the elements and attributes in an HDF4 mapping file. These elements and attributes are used to describe the data and grouping objects in HDF4 files, as well as the structural and application metadata associated with those objects and with the entire HDF4 file—the Level I and Level II information discussed earlier in Section 1.4.

The schema is also used to validate that a specific XML document conforms to the HDF4 mapping schema, and is therefore a valid HDF4 mapping file. The W3C XML Schema language was used to express the prototype HDF4 mapping schema.

Figure 6 is a graphical representation of the prototype HDF4 mapping file schema. The individual elements in the schema, and their relationships to the HDF4 file contents, are discussed in Sections 2.2.1-2.2.9. Each section contains a list of the XML attributes for the element(s) being discussed. In those XML attribute lists, the symbol □ appears before optional attributes, and the symbol ☒ appears before required attributes.

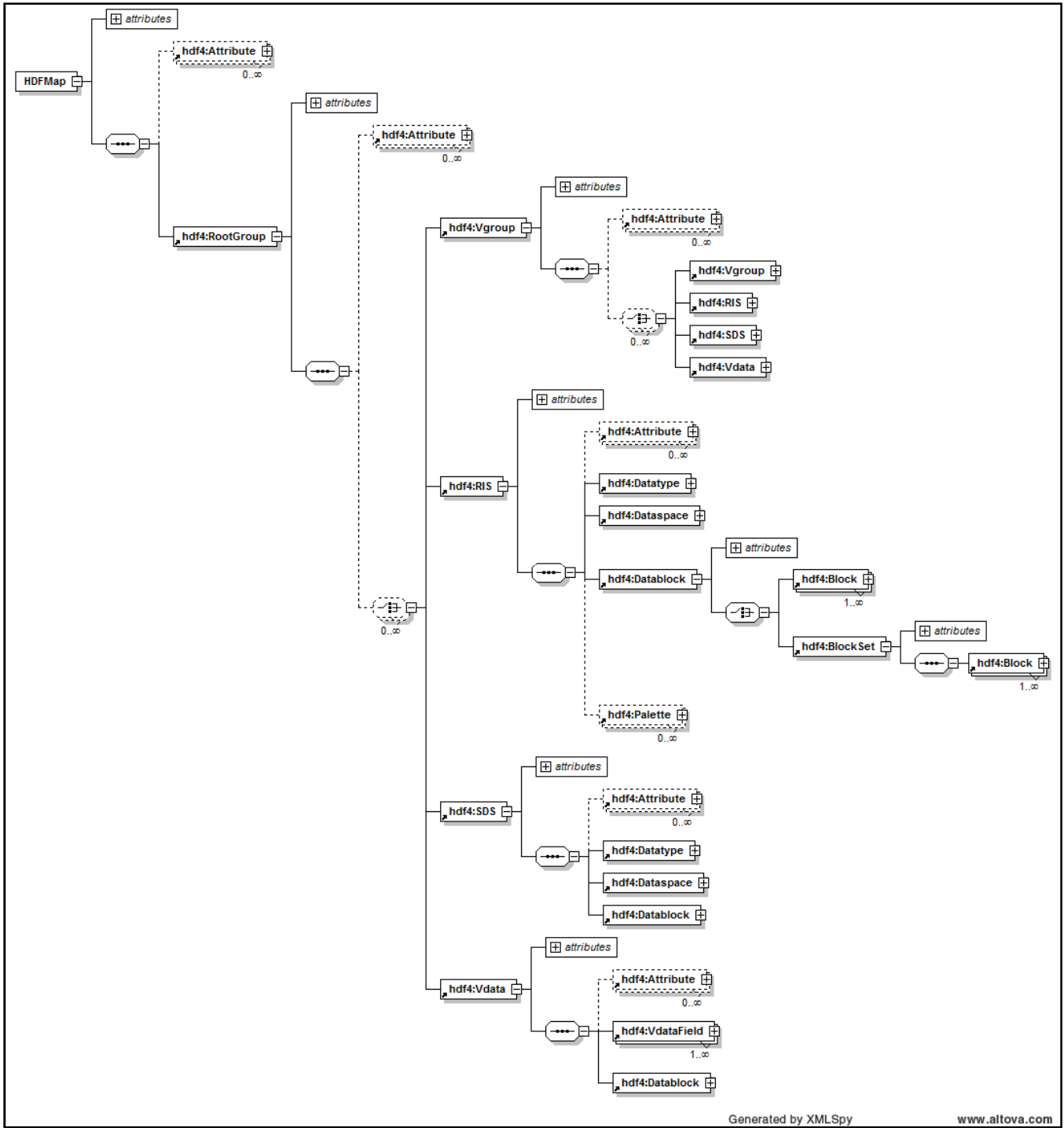


Figure 6 – Graphical representation of the prototype HDF4 mapping file schema.

## 2.2.1 HDFMap Element

A prototype HDF4 mapping file is an XML document with the root element *HDFMap*. The HDFMap element does not correspond directly to any information in the HDF4 file.

HDFMap may contain zero or more *Attribute* subelements (Section 2.2.2). Although the schema allows *Attribute* subelements in the HDFMap element, they are not used in the current implementation.

HDFMap has one required subelement, *RootGroup* (Section 2.2.3).

Finally, HDFMap has three optional attributes:

- *srcFile*: The name of the HDF4 file that the mapping file was created from.
- *srcVersion*: The version of the library that created the HDF4 file.
- *srcMd5sum*: The checksum generated by MD5 for digital signing and verification. The checksum can be used to determine if the HDF4 data file was modified after the mapping file was created.

The schema, in the XML Schema language, and the graphical representation of the HDFMap element are shown in Figure 7.

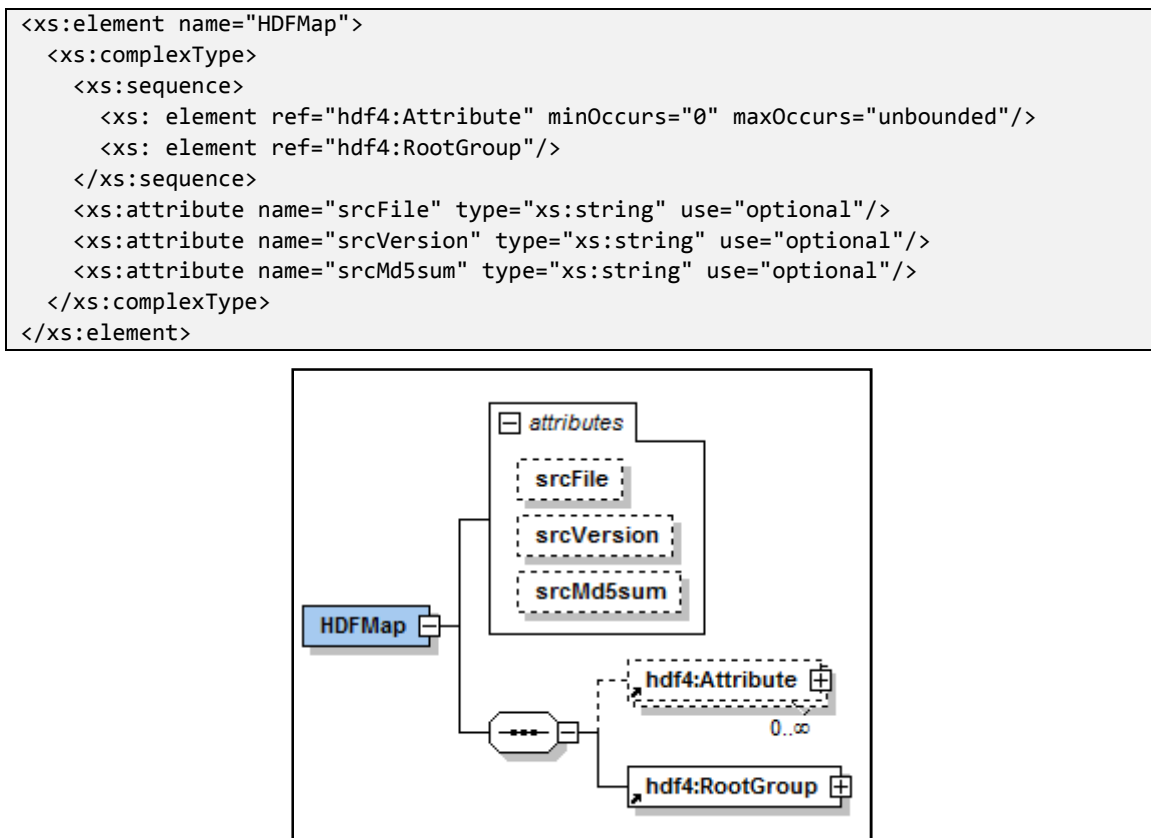


Figure 7 – HDFMap, the HDF4 mapping file root element.

## 2.2.2 Attribute Element

---

XML *Attribute elements*, not to be confused with XML *attributes*, are used to store *HDF4 attributes*. The HDF4 attributes hold the *application metadata*. *HDF4 annotations* also hold application metadata in HDF4 files, but *HDF4 annotations are not represented in the prototype mapping files*.

HDF4 includes some predefined attributes that could be considered *structural metadata*, because their meaning is shared by multiple applications when interpreting the object data. For the purposes of the HDF4 mapping schema, the predefined attributes are represented as Attribute elements, along with the application metadata. See Section 3.10 in the HDF4 User's Guide for more details on the predefined attributes.

As shown in Figure 6, Attribute elements can occur as subelements of many different elements in an HDF4 Mapping file. Their position in the mapping file reflects where the attributes they represent occur in the original HDF4 file.

Attribute subelements in the RootGroup element of the mapping file are used to store HDF4 global attributes—the application metadata that applies to the entire HDF4 file. Attribute subelements in the Vgroup, RIS, SDS, and Vdata elements (Sections 2.2.4, 2.2.5, 2.2.6, and 2.2.7) are used to store the HDF4 attributes associated with the corresponding HDF4 grouping and data objects.

Each HDF4 attribute has a *name*, *datatype*, and *value*. In an HDF4 mapping file, all of the information for a given HDF4 attribute is stored in a single XML Attribute.

An Attribute element has one required and one optional XML attribute. *In the prototype HDF4 mapping files, all Attribute elements always have both XML attributes:*

- ☒ *name*: The name of the HDF4 attribute this Attribute element corresponds to. One example for a predefined attribute is `name="valid_range"`.
- ☒ *ntDesc*: The description of the “number type” for the HDF4 attribute value. This may in fact describe a numeric *or* a character type. For example, `ntDesc="8-bit unsigned integer"` or `ntDesc="8-bit signed char"`.

As noted earlier, an HDF4 Attribute element is different than an XML attribute. An XML attribute describes a property of an XML element. An XML attribute is part of an XML element. An Attribute element is an XML element and contains XML attributes.

The “content” of the Attribute element is the value of the corresponding HDF4 attribute, which can be interpreted based on the type specified by *ntDesc*. If the type is a numeric one, there may be multiple numbers in the element content, separated by whitespace. Since the Attribute element stores application metadata, it is up to the application to parse the element content correctly.

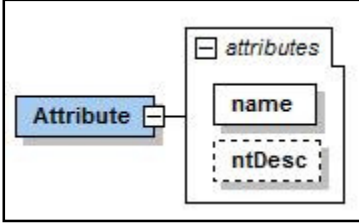
An Attribute element schema, diagram, and two instance examples are shown in Figure 8. Since the instance examples are shown without the context of the entire HDF mapping file document, it is not possible to tell what data objects they refer to.

The first Attribute element instance example has two XML attributes: (1) `name="valid_range"`, and (2) `ntDesc="8-bit unsigned integer"`. This Attribute element corresponds to an HDF4 predefined attribute (`valid_range`). The value of the HDF4 attribute, which is the content of the XML element, is `"0 255"`. This value should be interpreted by the application as two 8-bit unsigned integers, the low and high ends of the valid range.

The second instance example has `name="MOD10InputGranuleNames"` and `ntDesc="8-bit signed char"`. This Attribute element corresponds to HDF-EOS metadata; Level III information represented as

Level II HDF4 application metadata. The value of the HDF4 attribute is the string that begins "MOD10\_L2.A20070...". This value would have meaning to the HDF-EOS library.

```
<xs:element name="Attribute">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="ntDesc" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
```



```
< hdf4:Attribute name="valid_range" ntDesc="8-bit unsigned integer">
  0 255
</ hdf4:Attribute >
```

```
< hdf4:Attribute name="MOD10InputGranuleNames" ntDesc="8-bit signed char">
  MOD10_L2.A2007001.0000.004.2007001210207.hdf,MOD10_L2.A2007001.1950.004.2
  007002085622.hdf,MOD10_L2.A2007001.2130.004.2007002091535.hdf,MOD10_L2.A200
  7001.2305.004.2007002132656.hdf
</ hdf4:Attribute >
```

Figure 8 – Attribute element: schema, diagram, and two instance examples.

### 2.2.3 RootGroup Element

The *RootGroup* element in an HDF4 mapping file represents the *root* of the object hierarchy in an HDF4 file.

The *RootGroup* element can have zero or more Attribute subelements, followed by zero or more Vgroup, RIS, SDS, and Vdata subelements (in any order).

The *RootGroup* element has two required XML attributes:

- ☒ *objName*: This attribute has a fixed value of `objName="/"`.
- ☒ *objID*: This attribute has a fixed value of `objID="xid_0_0"`.

The schema and the graphical representation of the *RootGroup* element are shown in Figure 9.

```
<xs:element name="RootGroup">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="hdf4:Attribute" minOccurs="0" maxOccurs="unbounded"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="hdf4:Vgroup"/>
        <xs:element ref="hdf4:RIS"/>
        <xs:element ref="hdf4:SDS"/>
        <xs:element ref="hdf4:Vdata"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="objName" type="xs:string" fixed="/" />
    <xs:attribute name="objID" type="xs:string" fixed="xid_0_0" />
  </xs:complexType>
</xs:element>
```

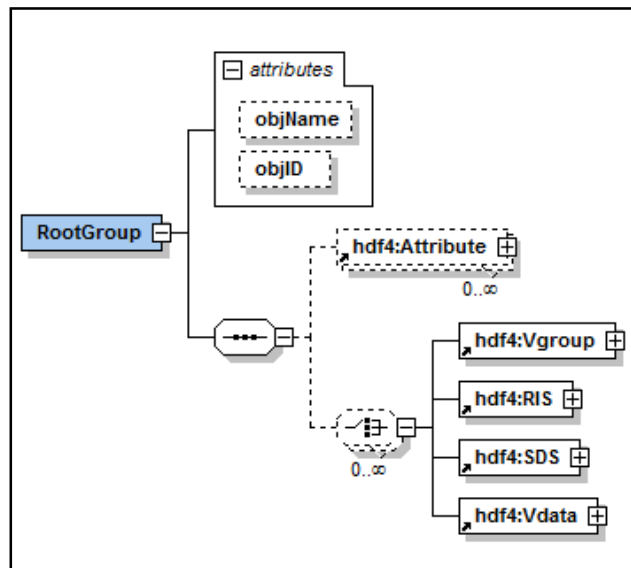


Figure 9 – *RootGroup* element.

## 2.2.4 Vgroup Element

An XML *Vgroup* element represents an *HDF4 Vgroup*—a grouping object.

A *Vgroup* element can have zero or more *Attribute* subelements, followed by zero or more *Vgroup*, *RIS*, *SDS*, and *Vdata* subelements (in any order).

A *Vgroup* element has three required XML attributes:

- ☒ *objName*: The name of the corresponding *Vgroup* object in the HDF4 file. For example, `objName="Data Fields"`.
- ☒ *objPath*: The path to the corresponding *Vgroup* object in the HDF4 file. The path does not include the name of the *Vgroup*. For example, `objPath="/MOD_Grid_Snow_500m"`.
- ☒ *objID*: A string that uniquely identifies the element in the HDF4 mapping file. The *objID* begins with "xid\_", followed by a combination of the *Tag* and *Reference Number* of the corresponding HDF4 *Vgroup* object. For *Vgroup* objects, the tag is always `DFTAG_VG`. See the *HDF4 User's Guide* for more information on HDF4 Tags and Reference Numbers. An example is: `objID="xid_DFTAG_VG-3"`.

The schema and the graphical representation of the *Vgroup* element are shown in Figure 10.

```
<xs:element name="Vgroup">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="hdf4:Attribute" minOccurs="0" maxOccurs="unbounded"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="hdf4:Vgroup"/>
        <xs:element ref="hdf4:RIS"/>
        <xs:element ref="hdf4:SDS"/>
        <xs:element ref="hdf4:Vdata"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="objName" type="xs:string" use="required"/>
    <xs:attribute name="objPath" type="xs:string" use="required"/>
    <xs:attribute name="objID" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

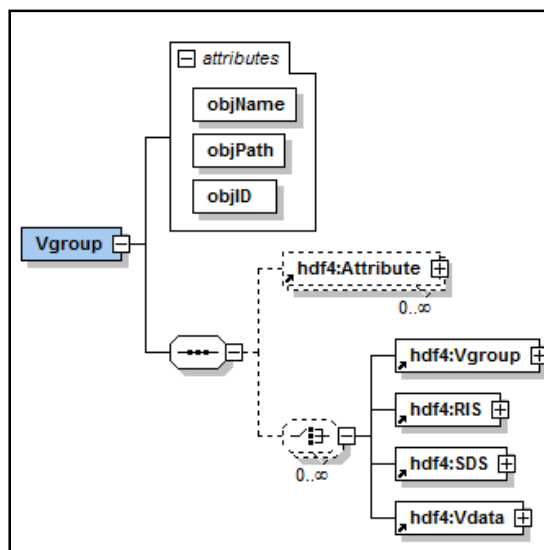


Figure 10 – Vgroup element.

### RIS Element

An XML *RIS* element represents an *HDF4 raster image*. While the HDF4 data format supports 8-bit, 24-bit, and general raster images, a single RIS element in the prototype HDF4 mapping file is designed to represent any of the three raster image types supported by HDF4. *The prototype hmap only handles 8-bit raster images*. Refer to the HDF4 User's Guide for further details on how raster images are represented in HDF4.

A raster image is a data object and has Level I information—object data—associated with it. As explained in Section 1.4.1, the object data remains in the binary HDF4 file. The mapping file contains the information needed to retrieve the raster image object data from the HDF4 file without calling the HDF4 library.

An RIS element can have zero or more Attribute subelements followed by a Datatype subelement, a Dataspace subelement, and a Datablock subelement. The Datatype and Dataspace hold structural metadata about the data type, byte order, and number and size of dimensions of the object data. The Datablock holds the location and compression information needed to retrieve the object data from the HDF4 binary file. These elements are discussed in Sections 2.2.8 and 2.2.9. Following the Datatype, Dataspace, and Datablock subelements, the RIS element may contain zero or more Palette subelements, which are detailed later in this section.

An RIS element has three required and two optional XML attributes:

- objName*: The name of the corresponding raster image object in the HDF4 file.
- objPath*: The path to the corresponding raster image object in the HDF4 file. The path does not include the name of the raster image.
- objID*: A string that uniquely identifies the element in the HDF4 mapping file. The objID value begins with "xid\_", followed by a combination of the *Tag* and *Reference Number* of the corresponding HDF4 object. See the HDF4 User's Guide for more information on HDF4 Tags and Reference Numbers. For RIS elements, the tag will be DFTAG\_RI8 or DFTAG\_RIG.
- ncomp*: The number of components in the raster image. For example, `ncomp="3"` for an RGB image and `ncomp="4"` for a CMYK image.
- interlace*: The interlace mode to use when reading the data. Defaults to "PIXEL", with "LINE" and "PLANE" the other possible options. See the HDF4 User's Guide for more information.

### Palette Element

An XML *Palette* element represents an *HDF4 Palette* (i.e., a color lookup table). For the HDF4 mapping prototype, palette information is treated as Level-II information (structural metadata), and therefore is represented directly in the HDF4 mapping file.

A Palette element has one required and three optional XML attributes:

- nentries*: The number of entries in the palette.
- ncomp*: The number of components in the palette.
- interlace*: The interlace mode of the stored palette data. Defaults to "PIXEL", with "LINE" and "PLANE" the other possible options.
- ntDesc*: The description of the "number type" for the palette values. This may, in fact, describe a numeric or a character type. Defaults to "8-bit unsigned char".

The "content" of the Palette element is the palette (color table) entries, which can be interpreted based on the values of *ncomp*, *interlace*, and *ntDesc*. If the type (*ntDesc*) is a numeric one, as is typically the case, there will be multiple numbers in the element content, separated by whitespace.

The schema and the graphical representation of the RIS and Palette elements are shown in Figure 11.

```

<xs:element name="RIS">
<xs:complexType>
<xs:sequence>
<xs:element ref="hdf4:Attribute" minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="hdf4:Datatype"/>
<xs:element ref="hdf4:Dataspace"/>
<xs:element ref="hdf4:Datablock"/>
<xs:element ref="hdf4:Palette" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="objName" type="xs:string" use="required"/>
<xs:attribute name="objPath" type="xs:string" use="required"/>
<xs:attribute name="objID" type="xs:string" use="required"/>
<xs:attribute name="ncomp" type="xs:nonNegativeInteger" use="optional"/>
<xs:attribute name="interlace" type="hdf4:interlaceType" use="optional" default="PIXEL"/>
</xs:complexType>
</xs:element>

```

```

<xs:element name="Palette">
<xs:complexType>
<xs:attribute name="nentries" type="xs:nonNegativeInteger" use="required"/>
<xs:attribute name="ncomp" type="xs:nonNegativeInteger" use="optional"/>
<xs:attribute name="interlace" type="hdf4:interlaceType" use="optional" default="PIXEL"/>
<xs:attribute name="ntDesc" type="xs:string" use="optional" default="8-bit unsigned char"/>
</xs:complexType>
</xs:element>

```

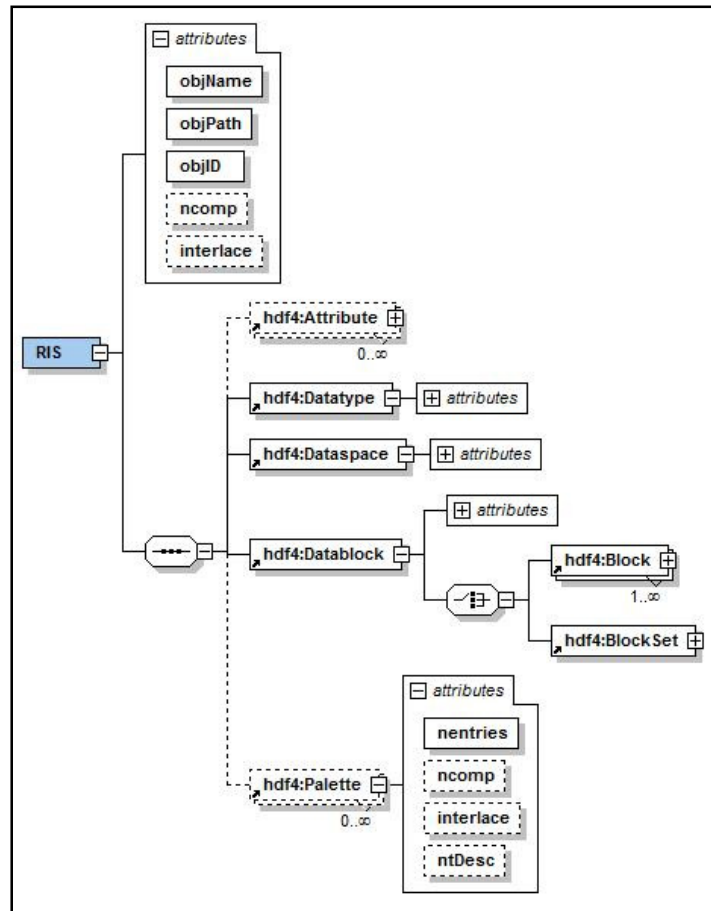


Figure 11 – RIS and Palette elements.

## 2.2.6 SDS Element

---

An XML *SDS* element represents an *HDF4 Scientific Data Set*. While the HDF4 data format supports the ability to store SDS arrays in external files, *the HDF4 mapping prototype does not support the external file functionality*.

A Scientific Data Set is a data object and has Level 1 information—object data—associated with it. As explained in Section 1.4.1, the object data remains in the binary HDF4 file. The mapping file contains the information needed to retrieve the Scientific Data Set object data without calling the HDF4 library.

An SDS element can have zero or more Attribute subelements followed by a Datatype subelement, a Dataspace subelement, and a Datablock subelement. The Datatype and Dataspace hold structural metadata about the data type, byte order, and number and size of dimensions of the object data. The Datablock holds the location and compression information needed to retrieve the object data from the HDF4 binary file. These elements are discussed in Sections 2.2.8 and 2.2.9.

An SDS element has three required XML attributes:

- ☒ *objName*: The name of the corresponding Scientific Data Set object in the HDF4 file.
- ☒ *objPath*: The path to the corresponding Scientific Data Set object in the HDF4 file. The path does not include the name of the Scientific Data Set object.
- ☒ *objID*: A string that uniquely identifies the element in the HDF4 mapping file. The objID value begins with "xid\_", followed by a combination of the *Tag* and *Reference Number* of the corresponding HDF4 object. See the HDF4 User's Guide for more information on HDF4 Tags and Reference Numbers. For SDS elements, the tag will be DFTAG\_SD, DFTAG\_SDG, or DFTAG\_NDG.

The schema and the graphical representation of the SDS element are shown in Figure 12 on the next page.

```

<xs:element name="SDS">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="hdf4:Attribute" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="hdf4:Datatype"/>
      <xs:element ref="hdf4:Dataspace"/>
      <xs:element ref="hdf4:Datablock"/>
    </xs:sequence>
    <xs:attribute name="objName" type="xs:string" use="required"/>
    <xs:attribute name="objPath" type="xs:string" use="required"/>
    <xs:attribute name="objID" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

```

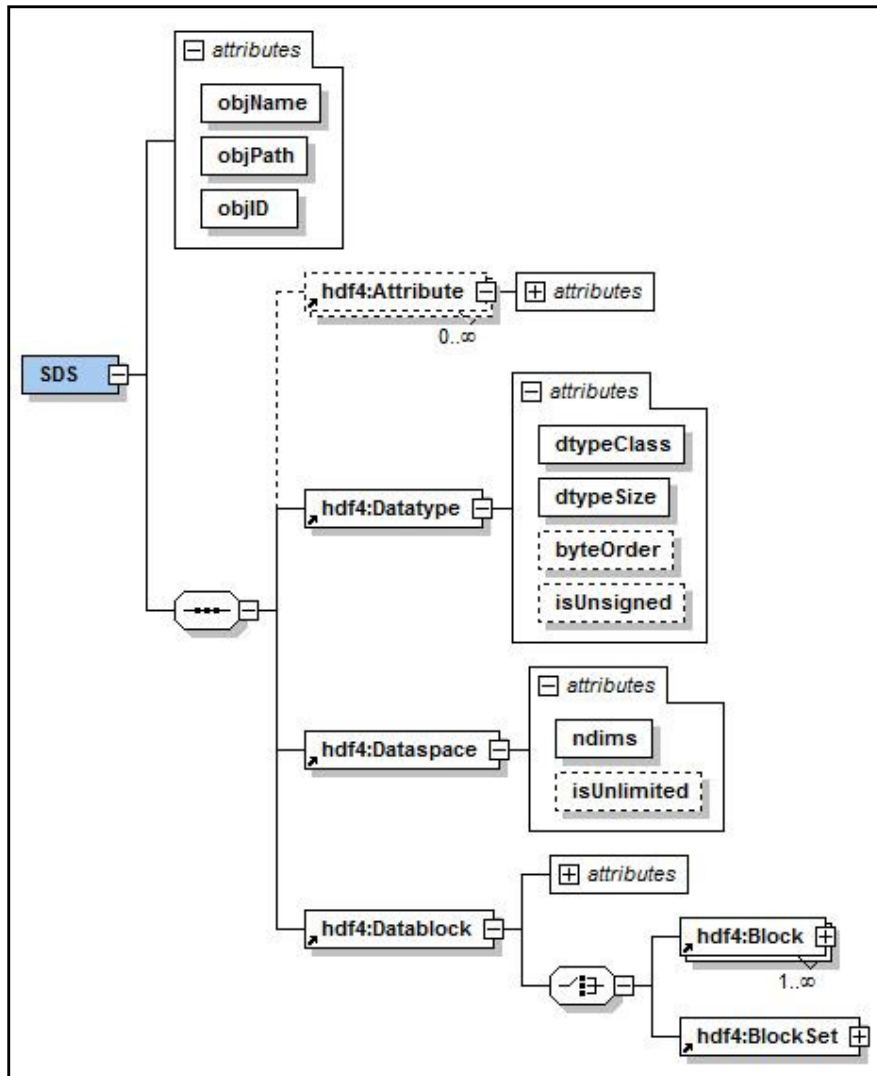


Figure 12 – SDS element.

### Vdata Element

An XML *Vdata* element represents an *HDF4 Vdata* object (a customized table).

A *Vdata* is a data object and has Level 1 information—object data—associated with it. As explained in Section 1.4.1, the object data remains in the binary HDF4 file. The mapping file contains the location information needed to retrieve the *Vdata* object data without calling the HDF4 library, in addition to structural and application metadata. *The prototype implementation does not capture all of the application metadata associated with the Vdata object, in particular Class is not in the mapping file.*

A *Vdata* element can have zero or more *Attribute* subelements, followed by one or more *VdataField* subelements, followed by a *Datablock* subelement. The *VdataField* element is detailed later in this section. The *Datablock* holds the location and compression information needed to retrieve the object data from the HDF4 binary file, and is discussed in Section 2.2.9.

A *Vdata* element has four required and three optional XML attributes:

- objName*: The name of the corresponding *Vdata* object in the HDF4 file.
- objPath*: The path to the *Vdata* object in the corresponding HDF4 file. The path does not include the name of the *Vdata* object.
- objID*: A string that uniquely identifies the element in the HDF4 mapping file. The *objID* value begins with "xid\_", followed by a combination of the *Tag* and *Reference Number* of the corresponding HDF4 object. See the HDF4 User's Guide for more information on HDF4 Tags and Reference Numbers. For *Vdata* elements, the tag will be DFTAG\_VS or DFTAG\_VH.
- nFields*: The number of fields (columns) in the *Vdata* table.
- nEntries*: The number of entries (rows) in the *Vdata* table.
- nBytes*: The number of bytes it takes to store an entry (row).
- interlaced*: A flag indicating how the data values are stored in the HDF4 file. When set to false (the default), all fields in the first row of the table are stored contiguously, followed by all fields in the second row, and so on. When set to true, the field 1 values for all rows are stored contiguously, followed by the field 2 values for all rows, and so on.

The schema for the *Vdata* element is shown in Figure 13. Figure 15 shows the graphical representation.

```
<xs:element name="Vdata">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="hdf4:Attribute" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="hdf4:VdataField" maxOccurs="unbounded"/>
      <xs:element ref="hdf4:Datablock"/>
    </xs:sequence>
    <xs:attribute name="objName" type="xs:string" use="required"/>
    <xs:attribute name="objPath" type="xs:string" use="required"/>
    <xs:attribute name="objID" type="xs:string" use="required"/>
    <xs:attribute name="nFields" type="xs:positiveInteger" use="required"/>
    <xs:attribute name="nEntries" type="xs:positiveInteger" use="optional"/>
    <xs:attribute name="nBytes" type="xs:positiveInteger" use="optional"/>
    <xs:attribute name="interlaced" type="xs:boolean" use="optional" default="false"/>
  </xs:complexType>
</xs:element>
```

Figure 13 – *Vdata* element schema.

### VdataField Element

An XML *VdataField* element represents a field (column) in an *HDF4 Vdata* (customized table) object. The *VdataField* contains Level II information; application and structural metadata.

A *VdataField* element can have zero or more Attribute subelements, followed by a *Datatype* subelement. The *Datatype* element is detailed in Section 2.2.8.

A *VdataField* element has one required and four optional XML attributes:

- name*: The name of the field.
- size*: The number of bytes it takes to store the field. This is equal to the order (see next attribute) multiplied by the number of bytes it takes to store a single value of this *Datatype*.
- order*: The number of values in the field. A field (column) in a *Vdata* table can store multiple values of the same data type. See the *HDF4 User's Guide* for more information about this capability.
- offset*: The position of the first byte in this field with respect to the beginning of a row in the table. The first field has an offset of 0.
- namelen*: The number of bytes it takes to store the name of this field.

In the *HDF4* mapping prototype the *size*, *order*, and *offset* optional attributes are always used. The *namelen* attribute is never used.

The schema of the *VdataField* element is shown in Figure 14. Figure 15 shows the graphical representation.

```
<xs:element name="VdataField">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="hdf4:Attribute" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="hdf4:Datatype"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="size" type="xs:positiveInteger" use="optional"/>
    <xs:attribute name="order" type="xs:nonNegativeInteger" use="optional"/>
    <xs:attribute name="offset" type="xs:nonNegativeInteger" use="optional"/>
    <xs:attribute name="namelen" type="xs:positiveInteger" use="optional"/>
  </xs:complexType>
</xs:element>
```

Figure 14 – *VdataField* element schema.

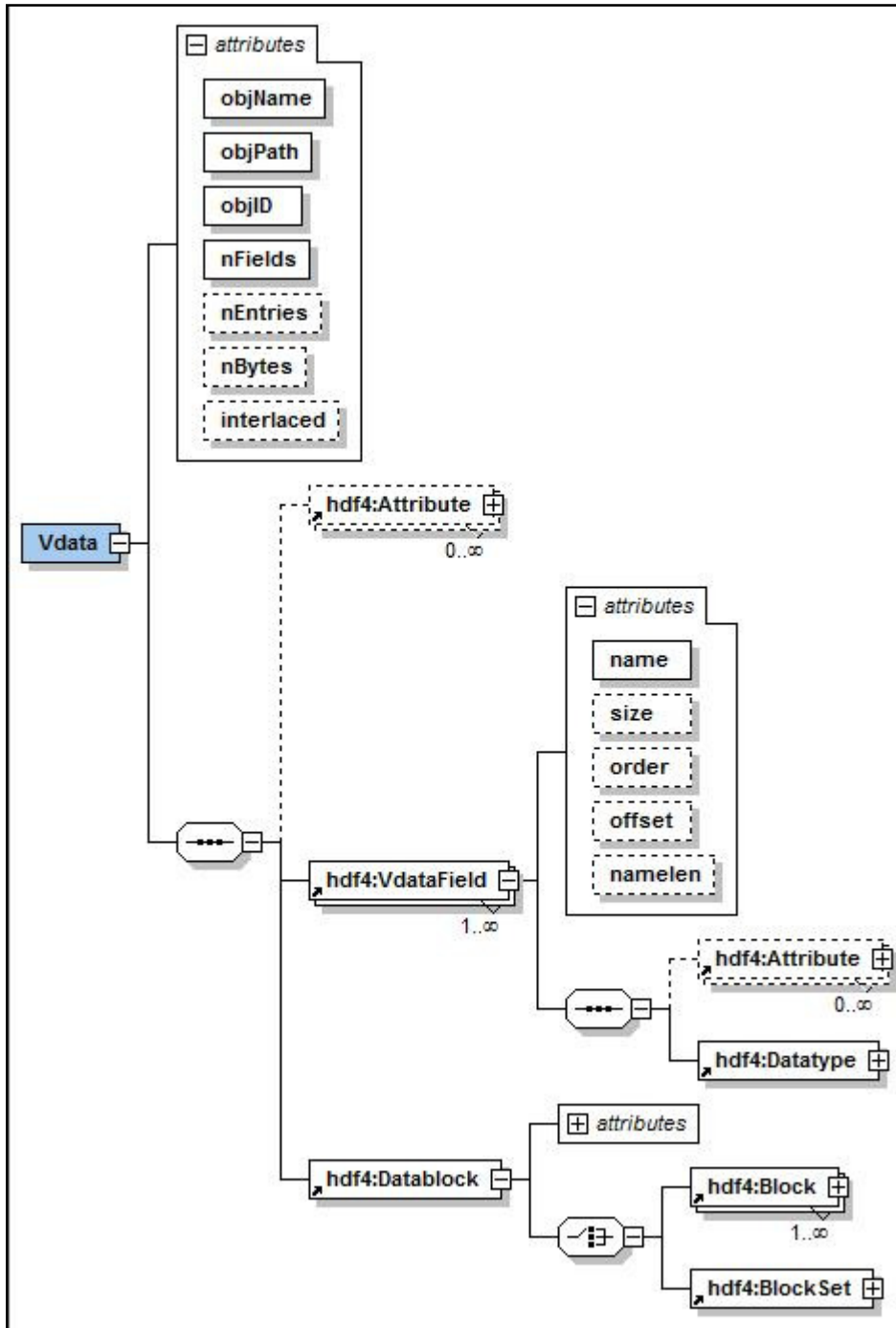


Figure 15 – Graphical representation of Vdata and VdataField elements.

### Datatype Element

An XML *Datatype* element stores the data type of the data values (object data) for a given HDF4 raster image, SDS, or Vdata data object. This structural metadata includes the type (int, float, char, string), size in bytes, endianness, and byte order.

A Datatype element has two required and two optional XML attributes. *In the HDF4 mapping prototype if the byteOrder optional attribute is not explicitly specified it should be treated as little endian, even though the schema does not specify a default value.*

- ☒ *dtypeClass*: A string that specifies the type of the data. Valid values are “INT”, “FLOAT”, “CHAR”, and “STRING”.
- ☒ *dtypeSize*: The number of bytes used to represent one value in the data object’s array.
- ☐ *endianType*: The endianness (byte order) used to store the data. Valid values are “LE” (little endian) and “BE” (big endian).
- ☐ *isUnsigned*: A flag indicating whether the data value is unsigned or signed. Valid values are “true” and “false”. Defaults to “false”, indicating a signed value.

Figure 16 shows the schema, graphical representation, and an instance example for the Datatype element.

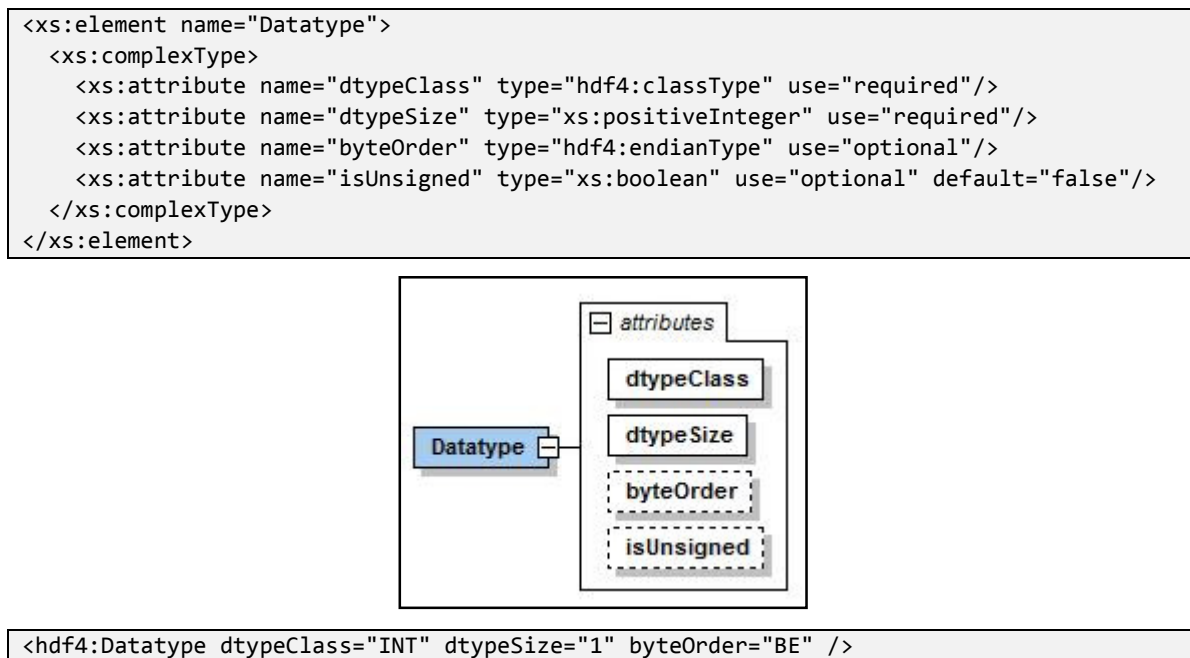


Figure 16 – Datatype element: schema, diagram, and instance example.

## Dataspace Element

An XML *Dataspace* element stores the number and size of dimensions for a given HDF4 raster image or SDS object data. This structural metadata is stored in the Dataspace element's XML attributes and content.

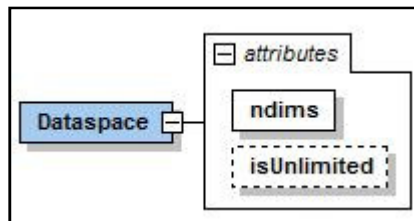
A Dataspace element has one required and one optional XML attributes.

- ndims*: The number of dimensions in the data object's array.
- isUnlimited*: A flag indicating whether the size of the array's first dimension can be extended. Valid values are "true" and "false". Defaults to "false", indicating the size of the first dimension is fixed.

The "content" of the Dataspace element is the dimension sizes, and should be interpreted as integer values separated by white space.

Figure 17 shows the schema, graphical representation, and an instance example for the Dataspace element.

```
<xs:element name="DataSpace">
  <xs:complexType>
    <xs:attribute name="ndims" type="xs:positiveInteger" use="required"/>
    <xs:attribute name="isUnlimited" type="xs:boolean" use="optional" default="false"/>
  </xs:complexType>
</xs:element>
```



```
<hdf4:Dataspace ndims="2"> 10 100 </hdf4:Dataspace>
```

Figure 17 – Dataspace element: schema, diagram, and instance example.

### 2.2.9 Datablock, Blockset, and Block Elements

---

The Datablock, Blockset, and Block elements in an HDF4 prototype mapping file collectively hold the locations of the object data in the original HDF4 file and the transformation information that is needed to retrieve the data from the HDF4 file. See Sections 1.4.1 and 1.5.

*The HDF4 mapping prototype schema does not strictly enforce the constraints based on the object data storage layout that are documented in this section. In the prototype implementation it is left up to the writer and reader to follow the documented conventions as an HDF4 mapping file is created and interpreted.*

#### Datablock Element

An XML *Datablock* element holds the location and compression information needed to retrieve object data for the corresponding RIS, Vdata, or SDS data object from the HDF4 binary file.

A Datablock element is organized in one of two ways depending on the storage layout of the object data in the HDF4 file (see Section 1.5).

- For *contiguous* and *chunked* storage layouts, the Datablock element will contain one or more *Block* subelement(s).
- For the *linked* storage layout, the Datablock element will contain one *BlockSet* subelement.

A Datablock element has two XML attributes; one required and one optional:

- nblocks*: The number of Block subelements in this Datablock element for contiguous and chunked storage layouts. Or, the number of Blocks in the Blockset subelement of this Datablock element for linked storage layout. Corresponds to the number of blocks or chunks used to store the object data in the HDF4 file.
- blockShape*: The chunk size in the HDF4 file for an object stored with the chunked storage layout. The *blockShape* attribute should contain a size for each dimension in the chunk, and the number of dimensions in the chunk should be the same as the number of dimensions in the data object array. For example, consider an HDF4 data object with three dimensions, dimension sizes [1024][65536][4], and a chunk size of [1024][256][4]. The corresponding Datablock element in the HDF4 mapping file would have the attribute `blockShape="1024x256x4"`.

#### Blockset Element

An XML *Blockset* element holds compression and location information when the linked storage layout is used for object data. If compression is specified, all the blocks of data in the Blockset must be uncompressed together (rather than individually).

The Blockset element contains one or more *Block* subelements. *Note that the prototype schema for the Blockset element encloses the Block subelement(s) in a sequence element. This is an artifact from an earlier version of the prototype schema, and is not required. Also note that Block subelements of the Blockset should not themselves specify compression.*

The Blockset element has one optional XML attribute:

- compression*: A string representing the compression method and, if required for uncompression, the parameters used when the object data was stored. The entire set of data in the blockset should be uncompressed using the appropriate method and parameters after being read from the HDF4 file. *In the prototype implementation, the schema does not define the valid values for the compression string. See the compression attribute description for the Block element for valid values.*

### Block Element

An XML *Block* element corresponds to a block or chunk of object data (Section 1.5) in a binary HDF4 file. The Block element contains sufficient information to read the block or chunk from the binary file, to uncompress the data in the block (if needed), and to position the data at the correct location in the data object array.

A Block element has two required and two optional XML attributes. *Although not prohibited by the prototype mapping schema, Blocks that are subelements of Blocksets should not use the origin or compression attributes.*

- offset*: The offset of the first byte of the block or chunk from the beginning of the HDF4 file.
- nbytes*: The number of bytes used to store the block or chunk in the HDF4 file.
- origin*: For chunked storage layouts, the position of this chunk of data in the overall chunk array. If the attribute is specified, it has the form "(d1, d2, d3, ... dn)", for a data object with n dimensions. The origin of the overall chunk array is (0,0,0,...0). An example offset is: `offset="(0,1,0)"`. Also see Figure 19 in Section 2.3.1.
- compression*: A string representing the compression method and, if required for uncompression, the parameters used when the object data was stored. The data in the block should be uncompressed using the appropriate method and parameters after being read from the HDF4 file. *In the prototype implementation, the schema does not define the valid values for the compression string.* The attribute will be of the form:
  - `compression="coder_type=JPEG"`
  - `compression="coder_type=DEFLATE"`
  - `compression="coder_type=NBIT,nt=%d,sign_ext=%d,fill_one=%d,start_bit=%d,bit_len=%d"`
  - `compression="coder_type=SKPHUFF"`
  - `compression="coder_type=SZIP,pixels=%d,pixels_per_scanline=%d,mask=%d,bits_per_pixel=%d,pixels_per_block=%d"`

The schema and graphical representation for the Datablock, Blockset, and Block elements are shown in Figure 18 on the next page.

```

<xs:element name="Datablock">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="hdf4:Block" maxOccurs="unbounded"/>
      <xs:element ref="hdf4:BlockSet"/>
    </xs:choice>
    <xs:attribute name="nblocks" type="xs:nonNegativeInteger" use="required"/>
    <xs:attribute name="blockShape" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="BlockSet">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="hdf4:Block" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="compression" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="Block">
  <xs:complexType>
    <xs:attribute name="offset" type="xs:nonNegativeInteger" use="required"/>
    <xs:attribute name="nbytes" type="xs:nonNegativeInteger" use="required"/>
    <xs:attribute name="origin" type="xs:string" use="optional"/>
    <xs:attribute name="compression" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

```

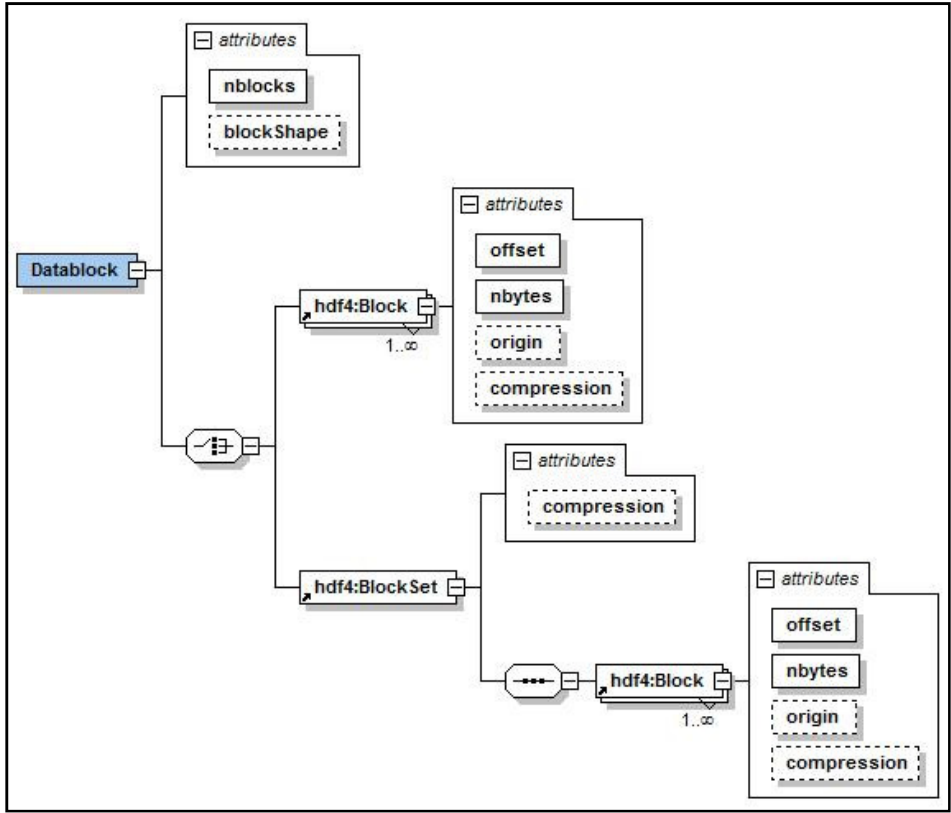


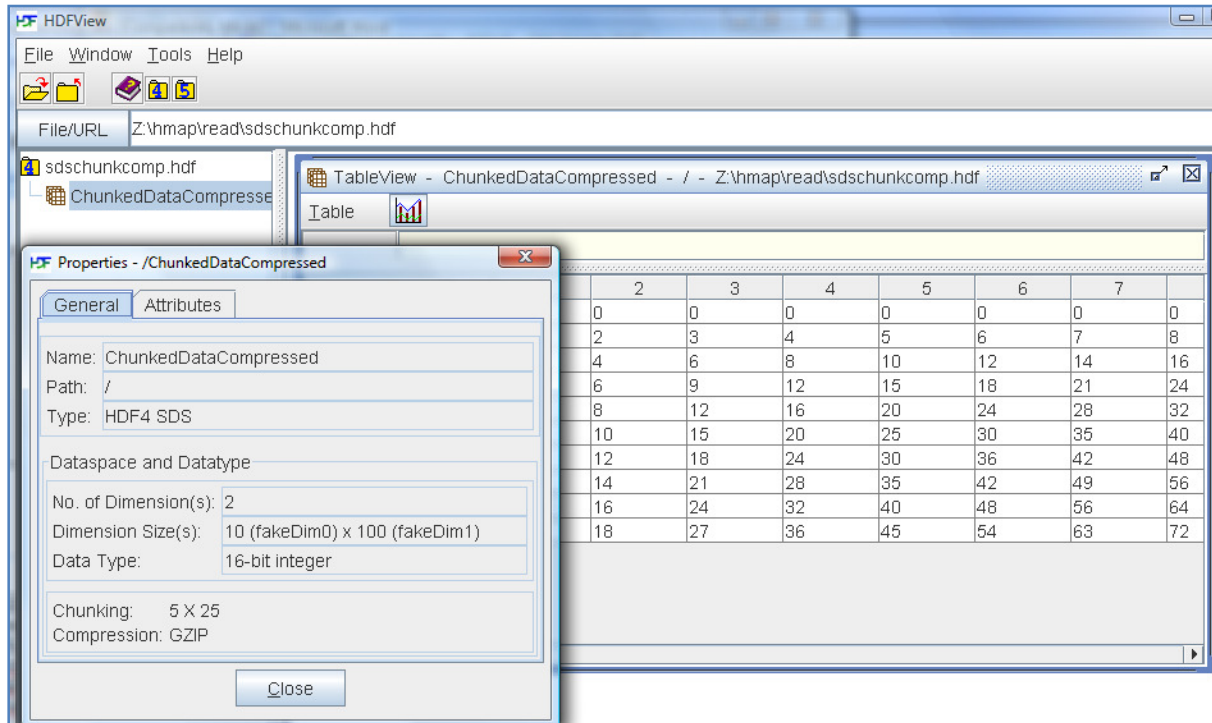
Figure 18 –Datablock, BlockSet, and Block elements.

## 2.3 Prototype HDF4 Mapping File Examples

A few examples of prototype HDF4 mapping files are given in this section. For each example, an *HDFView* screenshot shows a portion of the HDF4 binary file, followed by the HDF4 mapping file produced by *hmap*.

### 2.3.1 Scientific Data Set Example

Figure 20 shows a Scientific Data Set that uses the chunked storage layout with level 8 (the default level) DEFLATE compression. The object data array is (10x100) and there are 8 chunks of size (5x25).

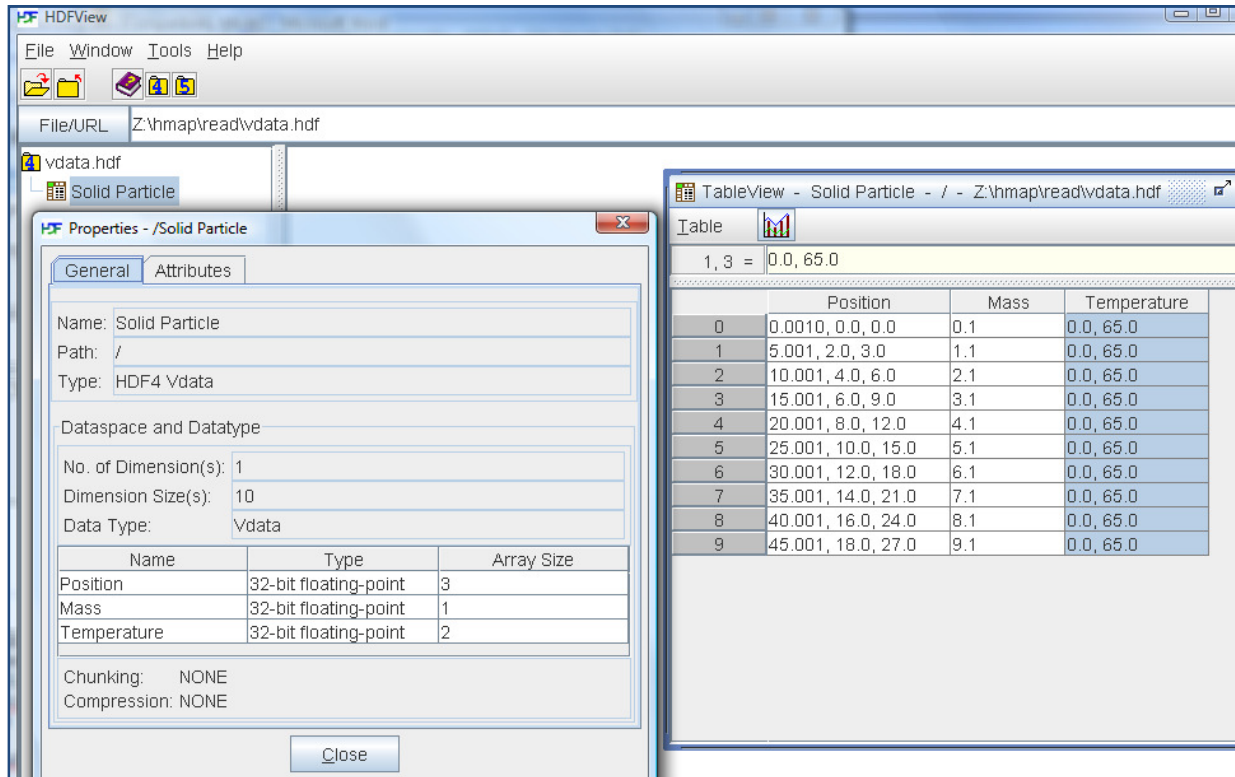


```
<?xml version="1.0" encoding="utf-8"?>
<hdf4:HDFMap xmlns:hdf4="http://www.hdfgroup.org/HDF4/HDF4Map">
  <hdf4:RootGroup>
    <hdf4:SDS objName="ChunkedDataCompressed" objPath="/" objID="xid-DFTAG_NDG-2">
      <hdf4:Datatype dtypeClass="INT" dtypeSize="4" byteOrder="BE" />
      <hdf4:Dataspace ndims="2">
        10 100
      </hdf4:Dataspace>
      <hdf4:Datablock nblocks="8" blockShape="5x25">
        <hdf4:Block offset="2609" nbytes="168" origin="(0,0)" compression="coder_type=DEFLATE" />
        <hdf4:Block offset="6939" nbytes="194" origin="(0,1)" compression="coder_type=DEFLATE" />
        <hdf4:Block offset="7149" nbytes="207" origin="(0,2)" compression="coder_type=DEFLATE" />
        <hdf4:Block offset="7372" nbytes="209" origin="(0,3)" compression="coder_type=DEFLATE" />
        <hdf4:Block offset="7597" nbytes="237" origin="(1,0)" compression="coder_type=DEFLATE" />
        <hdf4:Block offset="7850" nbytes="241" origin="(1,1)" compression="coder_type=DEFLATE" />
        <hdf4:Block offset="8107" nbytes="250" origin="(1,2)" compression="coder_type=DEFLATE" />
        <hdf4:Block offset="8373" nbytes="248" origin="(1,3)" compression="coder_type=DEFLATE" />
      </hdf4:Datablock>
    </hdf4:SDS>
  </hdf4:RootGroup>
</hdf4:HDFMap>
```

Figure 19 – Scientific Data Set with compression and chunked storage layout.

### 2.3.2 Vdata Example

Figure 20 shows a Vdata object with three fields. The Position field is a 32-bit float and has three values in it. The Mass field has a single 32-bit float value. The Temperature field has two 32-bit float values. Contiguous storage is used without compression.



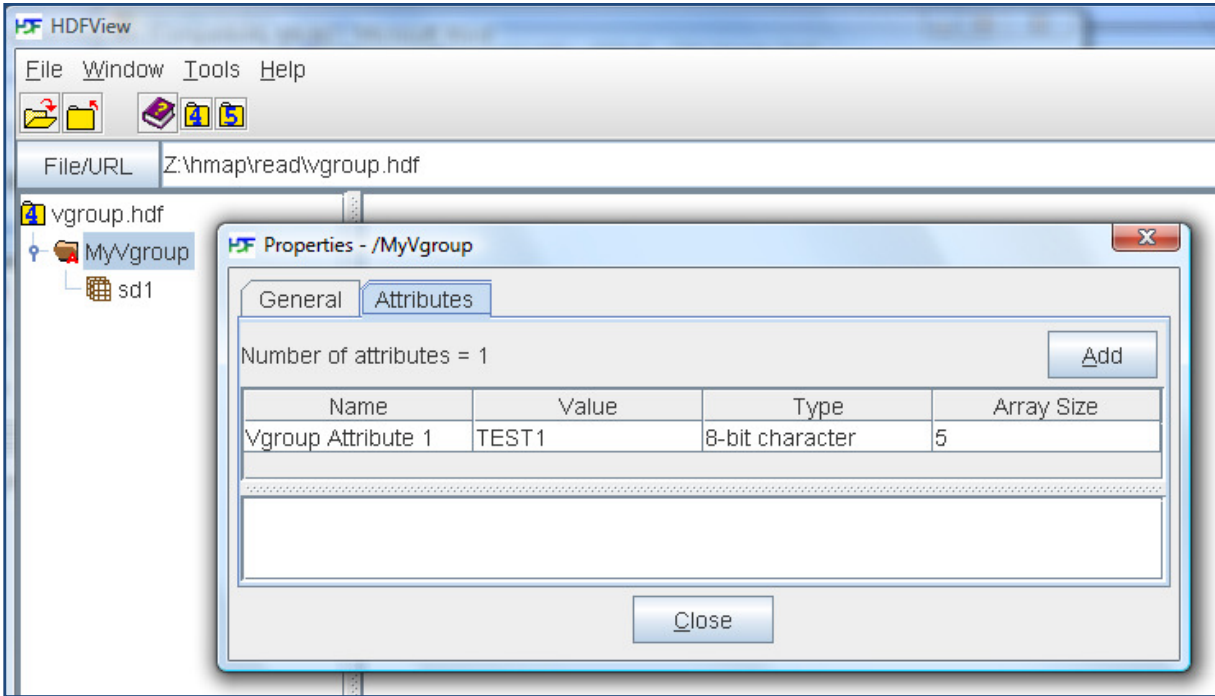
```
<?xml version="1.0" encoding="utf-8"?>
<hdf4:HDFMap xmlns:hdf4="http://www.hdfgroup.org/HDF4/HDF4Map">

  <hdf4:RootGroup>
    <hdf4:Vdata objName="Solid Particle" objPath="/" objID="xid-DFTAG_VS-2" nFields="3"
      nEntries="10" nBytes="24" interlaced="true">
      <hdf4:VdataField name="Position" size="12" order="3" offset="0">
        <hdf4:Datatype dtypeClass="FLOAT" dtypeSize="4" byteOrder="BE" />
      </hdf4:VdataField>
      <hdf4:VdataField name="Mass" size="4" order="1" offset="12">
        <hdf4:Datatype dtypeClass="FLOAT" dtypeSize="4" byteOrder="BE" />
      </hdf4:VdataField>
      <hdf4:VdataField name="Temperature" size="8" order="2" offset="16">
        <hdf4:Datatype dtypeClass="FLOAT" dtypeSize="4" byteOrder="BE" />
      </hdf4:VdataField>
      <hdf4:Datablock nblocks="1">
        <hdf4:Block offset="294" nbytes="240" />
      </hdf4:Datablock>
    </hdf4:Vdata>
  </hdf4:RootGroup>
</hdf4:HDFMap>
```

Figure 20 – Vdata with three fields, contiguous storage, and no compression.

### 2.3.3 Vgroup Example

Figure 21 shows a Vgroup object with one attribute element and one SDS element. The SDS is a one dimensional array with 10 elements. Contiguous storage is used without compression.



```
<?xml version="1.0" encoding="utf-8"?>
<hdf4:HDFMap xmlns:hdf4="http://www.hdfgroup.org/HDF4/HDF4Map">

  <hdf4:RootGroup>
    <hdf4:Vgroup objName="MyVgroup" objPath="/" objID="xid-DFTAG_VG-4">
      <hdf4:Attribute name="Vgroup Attribute 1" ntDesc="8-bit signed char">
        TEST1
      </hdf4:Attribute>
      <hdf4:SDS objName="sd1" objPath="/MyVgroup" objID="xid-DFTAG_NDG-2">
        <hdf4:Datatype dtypeClass="INT" dtypeSize="4" byteOrder="BE" />
        <hdf4:Dataspace ndims="1">
          10
        </hdf4:Dataspace>
        <hdf4:Datablock nblocks="1">
          <hdf4:Block offset="294" nbytes="40" />
        </hdf4:Datablock>
      </hdf4:SDS>
    </hdf4:Vgroup>
  </hdf4:RootGroup>
</hdf4:HDFMap>
```

Figure 21 – Vgroup with one attribute element and one SDS element.

## Acknowledgements

---

This work was supported by a Cooperative Agreement with the National Aeronautics and Space Administration (NASA) under NASA grant NNX06AC83A. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NASA.

Hampapuram K. Ramapriyan and Dan Marinelli, both from NASA, offered the project invaluable guidance and support.

Ruth E. Duerr, with the National Snow and Ice Data Center in Boulder Colorado, and Christopher Lynnes, with NASA in Greenbelt Maryland, were our project collaborators from the NASA EOSDIS data centers. Jonathan Crider, an undergraduate student at the University of Colorado, Boulder, also worked on the project. In addition to contributing to the definition of the prototype mapping schema, Ruth, Chris, and Jonathan were primarily responsible for conducting the survey of NASA's HDF4-formatted data, and for coding the reader programs.

Mike Folk, MuQun Yang, and Elena Pourmal, our colleagues at The HDF Group, participated in the design of the prototype mapping schema, handled project oversight, and provided HDF4 expertise.

Peter Cao was the primary developer of the prototype mapping schema and the *hmap* mapping file writer. Ruth Ayt contributed to the final version of this design and schema document.

## Further Information

---

For additional information about the project, please see <http://www.hdfgroup.org/projects/hdf4mapping> or contact the authors directly.